Single Position Signal Trading Simulation System

May 1, 2018

```
In [1]: %pylab inline

Populating the interactive namespace from numpy and matplotlib
```

1 Single Position Signal Trading Simulation System

1.1 Overview of libraries

The code below is meant to give an impression of the current implementation of the libraries that are customly developed for the algorithmic trading algorithm.

1.1.1 Imports

```
In [2]: import poloniex as plnx
        import ta_lib as ta
        from datetime import datetime, timedelta
        from matplotlib.finance import candlestick2_ohlc
```

1.1.2 Chart settings

```
In [3]: pair = 'USDT_BTC'
        timeframe = 30*60
        end = datetime.utcnow()
        start = end - timedelta(days=4)

In [4]: chart = plnx.get_chart(pair, timeframe, start, end)
```

1.1.3 Technical indicators

```
In [5]: ta.daily_returns(chart)
        ta.ema(chart)    # default n=7
        ta.ema(chart, 23)
        ta.bbands(chart) # default n=20
        ta.rsi(chart)    # default n=14
```

1.1.4 Plotting

In [6]:
```
from matplotlib import gridspec

gs = gridspec.GridSpec(2, 1, height_ratios=[3,1])
f = plt.figure(figsize=(20,15))
ax1 = plt.subplot(gs[0])
ax2 = plt.subplot(gs[1])
candlestick2_ohlc(ax1, chart['open'], chart['high'], chart['low'], chart['close'], widt
ax1.plot(chart['ema7'], c='lime', lw=2)
ax1.plot(chart['ema23'], c='red', lw=2)
ax1.plot(chart['bb_mid'], c='gray', ls='dotted')
ax1.plot(chart['bb_upper'], c='gray', ls='dotted')
ax1.plot(chart['bb_lower'], c='gray', ls='dotted')
ax1.grid()
ax2.plot(list(chart.index), chart['rsi'], c='gray')
ax2.grid()
ax2.axhline(y=80, c='red', ls='dashed')
ax2.axhline(y=20, c='lime', ls='dashed')
ax2.fill_between(list(chart.index), 20, 80, color='grey', alpha=0) # 'kinda' corrects
f.subplots_adjust(hspace=0)
```

1.1.5 Data columns

```
In [7]: chart.columns
```

```
Out[7]: Index(['close', 'date', 'high', 'low', 'open', 'daily_return', 'ema7', 'ema23',
               'bb_mid', 'bb_upper', 'bb_lower', 'rsi'],
              dtype='object')
```

1.1.6 Dataset

```
In [8]: chart[20:25]
```

```
Out[8]:           close                 date         high          low         open  \
        20  9098.169080  2018-04-26 23:30:00  9175.000000  9080.000000  9168.773684
        21  9159.143633  2018-04-27 00:00:00  9170.017888  9098.169080  9098.169080
        22  9225.341376  2018-04-27 00:30:00  9230.000000  9128.334587  9159.143633
        23  9244.796425  2018-04-27 01:00:00  9270.000000  9189.599015  9222.000000
        24  9286.000000  2018-04-27 01:30:00  9299.000000  9197.098010  9232.605917

            daily_return         ema7        ema23       bb_mid     bb_upper  \
        20    -70.604604  9074.742928          NaN  8958.288332  9192.056089
        21     60.974552  9108.188311          NaN  8982.224883  9241.774208
        22     66.197743  9152.517190  8995.344645  9010.870418  9304.396245
        23     19.455049  9190.065537  9021.857784  9039.544022  9361.818619
        24     41.203575  9222.921864  9049.786140  9070.695253  9418.297899

               bb_lower        rsi
        20  8724.520576  70.430204
        21  8722.675557  76.191020
        22  8717.344591  80.589150
        23  8717.269425  81.390225
        24  8723.092606  83.455062
```

1.2 Specification

Specifications for the trading simulator.

1.2.1 Simulation parameters

We want to be able to adjust the following parameters for the simulation:

1. Symbol
2. Timeframe
3. Stop loss strategy: `none`, `limit`, `trail`
4. Stop loss percentage (max losses before the trade is stopped out)
5. Amount of leverage used
6. Fee percentage per order
7. Initial balance
8. Lot sizing
9. Signals, when to go long and short

1.2.2 Simulation metrics

We want to able to track the following metrics:

1. Total trades
2. Winning trades
3. Losing trades
4. Win/loss ratio
5. Trades stopped out
6. Fee paid
7. Fee paid (cumulative)
8. Gains
9. Gains (cumulative)
10. Drawdown
11. Max drawdown
12. Risk/reward ratio
13. Risk/reward average

1.2.3 Simulation types

We want to support the following simulation types:

1. Single simulation
2. Multiple simulations for parameter tuning

1.3 Implementation

1.3.1 Signals

The generated signals are appended to the chart dataset. There will be four columns:

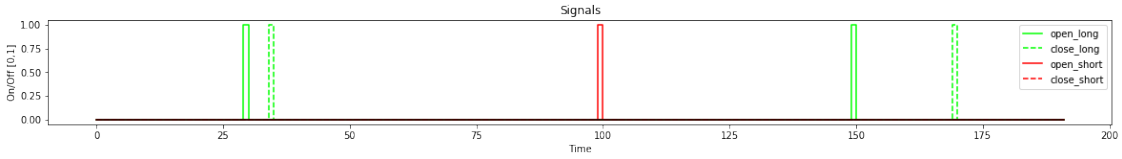
1. `open_long`
2. `close_long`
3. `open_short`
4. `close_short`

A value is 0 if the signal is off and 1 if the signal is on.

```
In [14]: chart['open_long'] = chart['close_long'] = chart['open_short'] = chart['close_short']

In [15]: chart.head(1)

Out[15]:          close                 date         high          low     open  \
         0  8850.110679  2018-04-26 13:30:00  8875.874887  8826.906467  8867.08

            daily_return  ema7  ema23  bb_mid  bb_upper  bb_lower  rsi  open_long  \
         0           NaN   NaN    NaN     NaN       NaN       NaN  NaN          0

            close_long  open_short  close_short
         0           0           0            0
```

```
In [37]: chart.loc[30, 'open_long'] = 1
         chart.loc[35, 'close_long'] = 1
         chart.loc[100, 'open_short'] = 1
         chart.loc[150, 'open_long'] = 1
         chart.loc[170, 'close_long'] = 1
```

```
In [40]: plt.figure(figsize=(20,2))
         step(list(chart.index), chart['open_long'], color='lime')
         step(list(chart.index), chart['close_long'], color='lime', linestyle='dashed')
         step(list(chart.index), chart['open_short'], color='red')
         step(list(chart.index), chart['close_short'], color='red', linestyle='dashed')
         step(list(chart.index), np.zeros(len(chart.index)), color='black')
         plt.title('Signals')
         plt.ylabel('On/Off [0,1]')
         plt.xlabel('Time')
         plt.legend(['open_long', 'close_long', 'open_short', 'close_short']);
```



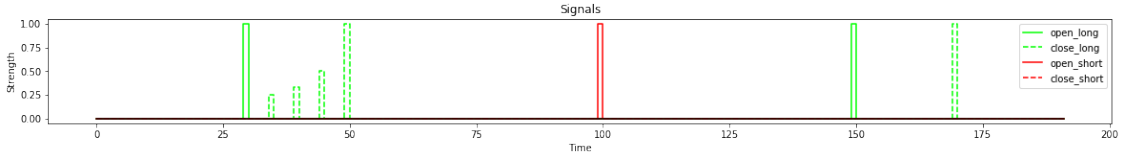
Perhaps let `close_long` and `close_short` indicate when we want to reduce a position. The interval used will be $[0, 1]$ where 0.5 will reduce the position by half, and 0.25 by a quarter.

Example: we have a position of 1000. If we want close this position by reducing with four orders, thus reduce 250 each time, we calculate them as follows:

1. `0.25 * order_size` for the first order to sell a quarter.
2. `0.33.. * order_size` for the next quarter (taking into account that the `order_size` has became smaller, but we still want to sell the same position size.)
3. `0.5 * order_size` for the third quarter.
4. `1.0 * order_size` for the last quarter.

```
In [45]: chart.loc[35, 'close_long'] = 0.25
         chart.loc[40, 'close_long'] = 1/3
         chart.loc[45, 'close_long'] = 0.50
         chart.loc[50, 'close_long'] = 1
```

```
In [48]: plt.figure(figsize=(20,2))
         step(list(chart.index), chart['open_long'], color='lime')
         step(list(chart.index), chart['close_long'], color='lime', linestyle='dashed')
         step(list(chart.index), chart['open_short'], color='red')
         step(list(chart.index), chart['close_short'], color='red', linestyle='dashed')
         step(list(chart.index), np.zeros(len(chart.index)), color='black')
         plt.title('Signals')
         plt.ylabel('Strength')
         plt.xlabel('Time')
         plt.legend(['open_long', 'close_long', 'open_short', 'close_short']);
```



1.3.2 Trade

We want a `Trade` instance that keeps track of:

- State (`open`, `closed`, `stopped`, `submitted`, `timed_out`, `expired`)
- Expiry time, if an order expires we want to resubmit a limit order for a new price.
- Lot size
- Entry price
- Exit price
- P/L
- P/L in %
- Stop loss type (`none`, `limit`, `trail`)
- Stop loss percentage
- Stop loss price
- Fees
- Leverage
- Net result (P/L adjusted with fees)
- Entry date
- Exit date
- Trade open (hours)

```
In [9]: class Trade():
            exit_price = None      # Price when the trade was closed.
            stop_loss_price = None # Price when the stop loss should trigger.

            """ Initialize a new order. """
            def __init__(self, position_type, lot_size, leverage, entry_price, stop_loss, stop_
                self.state = 'open'
                self.position_type = position_type
                self.lot_size_base = lot_size
                self.lot_size_base_leveraged = lot_size * leverage
                self.lot_size_quote = lot_size / entry_price
                self.lot_size_quote_leveraged = lot_size / entry_price * leverage
                self.leverage = leverage
                self.entry_price = entry_price
                self.stop_loss = stop_loss
                self.stop_loss_percentage = stop_loss_percentage
                self.fee_percentage = fee_percentage
                self.stop_loss_price = (1 - stop_loss_percentage) * entry_price if position_typ
                                       else (1 + stop_loss_percentage) * entry_price # is short
```

```
    """ Returns the total order cost in the base currency. """
    def order_cost(self):
        return self.lot_size_base * self.leverage

    """ Returns the market value of the order in the base currency. """
    def market_value(self):
        x = self.exit_price if self.exit_price != None else self.current_price
        return self.lot_size_quote * x * self.leverage

    """ Returns the P/L for this in the base currency. """
    def pl(self):
        x = self.exit_price if self.exit_price != None else self.current_price
        if self.position_type == 'long':  return self.market_value() - self.order_cost
        if self.position_type == 'short': return self.order_cost() - self.market_value

    """ Returns the P/L percentage. """
    def pl_percent(self):
        x = self.exit_price if self.exit_price != None else self.current_price
        if self.position_type == 'long':  return x / self.entry_price
        if self.position_type == 'short': return self.entry_price / x

    """ Returns the paid fees in the base currency. """
    def paid_fees(self):
        entry_fee = self.lot_size_base * self.leverage * self.fee_percentage
        exit_fee = self.lot_size_quote * self.leverage * (self.exit_price if self.exit_
        return entry_fee + exit_fee

    """ Closes the order on the current price. """
    def close(self):
        if self.state == 'stopped': return # order is already closed, and exit price i
                                           # for that stop loss level.
        self.exit_price = self.current_price
        self.state = 'closed'

    """ Closes the order on the stop loss price. """
    def stop(self):
        self.exit_price = self.stop_loss_price
        self.state = 'stopped'

    """ Update current price, this should be called before using any other method. """
    def update_price(self, current_price):
        self.current_price = current_price
        if self.stop_loss_hit():
            self.stop()

    """ Returns true if the stop loss price is hit. """
    def stop_loss_hit(self):
```

```
            if self.position_type == 'long':  return self.current_price < self.stop_loss_pr
            if self.position_type == 'short': return self.current_price > self.stop_loss_pr

        """ Statistical information. """
        def statistics(self):
            print('state:                    {}'.format(self.state))
            print('position_type:            {}'.format(self.position_type))
            print('lot_size_base:            {}'.format(self.lot_size_base))
            print('lot_size_base_leveraged:  {}'.format(self.lot_size_base_leveraged))
            print('lot_size_quote:           {}'.format(self.lot_size_quote))
            print('lot_size_quote_leveraged: {}'.format(self.lot_size_quote_leveraged))
            print('leverage:                 {}'.format(self.leverage))
            print('entry_price:              {}'.format(self.entry_price))
            print('exit_price:               {}'.format(self.exit_price))
            print('fee_percentage:           {}'.format(self.fee_percentage))
            print('stop_loss:                {}'.format(self.stop_loss))
            print('stop_loss_percentage:     {}'.format(self.stop_loss_percentage))
            print('stop_loss_price:          {}'.format(self.stop_loss_price))
            print('order_cost():             {}'.format(self.order_cost()))
            print('market_value():           {}'.format(self.market_value()))
            print('pl():                     {}'.format(self.pl()))
            print('pl_percent():             {}'.format(self.pl_percent()))
            print('paid_fees():              {}'.format(self.paid_fees()))
```

```
In [10]: position_type = 'long'
         lot_size = 10
         leverage = 10
         current_price = 500
         stop_loss = 'limit'
         stop_loss_percentage = 0.01
         fee_percentage = 0.00075

         trade1 = Trade(position_type, lot_size, leverage, current_price, stop_loss, stop_loss_
         trade1.update_price(750)
         trade1.close()
         trade1.statistics()
```

```
state:                    closed
position_type:            long
lot_size_base:            10
lot_size_base_leveraged:  100
lot_size_quote:           0.02
lot_size_quote_leveraged: 0.2
leverage:                 10
entry_price:              500
exit_price:               750
fee_percentage:           0.00075
stop_loss:                limit
```

```
stop_loss_percentage:     0.01
stop_loss_price:          495.0
order_cost():             100
market_value():           150.0
pl():                     50.0
pl_percent():             1.5
paid_fees():              0.1875
```

```
In [11]: position_type = 'short'
         lot_size = 1
         leverage = 1
         current_price = 1000
         stop_loss = 'limit'
         stop_loss_percentage = 0.01
         fee_percentage = 0.00075

         trade2 = Trade(position_type, lot_size, leverage, current_price, stop_loss, stop_loss_
         trade2.update_price(500)
         trade2.close()
         trade2.statistics()
```

```
state:                    closed
position_type:            short
lot_size_base:            1
lot_size_base_leveraged:  1
lot_size_quote:           0.001
lot_size_quote_leveraged: 0.001
leverage:                 1
entry_price:              1000
exit_price:               500
fee_percentage:           0.00075
stop_loss:                limit
stop_loss_percentage:     0.01
stop_loss_price:          1010.0
order_cost():             1
market_value():           0.5
pl():                     0.5
pl_percent():             2.0
paid_fees():              0.0011250000000000001
```

!! PL() should yield `1000.0`

```
In [12]: position_type = 'short'
         lot_size = 1000
         leverage = 1
         current_price = 1000
         stop_loss = 'limit'
```

```
        stop_loss_percentage = 0.01
        fee_percentage = 0.00075

        trade3 = Trade(position_type, lot_size, leverage, current_price, stop_loss, stop_loss
        trade3.update_price(1250)
        trade3.statistics()
```

```
state:                     stopped
position_type:             short
lot_size_base:             1000
lot_size_base_leveraged:   1000
lot_size_quote:            1.0
lot_size_quote_leveraged:  1.0
leverage:                  1
entry_price:               1000
exit_price:                1010.0
fee_percentage:            0.00075
stop_loss:                 limit
stop_loss_percentage:      0.01
stop_loss_price:           1010.0
order_cost():              1000
market_value():            1010.0
pl():                      -10.0
pl_percent():              0.9900990099009901
paid_fees():               1.5075
```

```
In [13]: position_type = 'short'
         lot_size = 1000
         leverage = 1
         current_price = 1000
         stop_loss = 'limit'
         stop_loss_percentage = 0.01
         fee_percentage = 0.00075

         trade4 = Trade(position_type, lot_size, leverage, current_price, stop_loss, stop_loss
         trade4.update_price(500)
         trade4.statistics()
```

```
state:                     open
position_type:             short
lot_size_base:             1000
lot_size_base_leveraged:   1000
lot_size_quote:            1.0
lot_size_quote_leveraged:  1.0
leverage:                  1
entry_price:               1000
exit_price:                None
```

```
fee_percentage:          0.00075
stop_loss:               limit
stop_loss_percentage:    0.01
stop_loss_price:         1010.0
order_cost():            1000
market_value():          500.0
pl():                    500.0
pl_percent():            2.0
paid_fees():             0.75
```

Add the ability to reduce positions, and calculate the fee individually (in a field `paid_fees`, instead of a function).

1.3.3 Single Position Signal Trading Simulation System

For simplicity, the trading simulation system will be only keep track of a single position. When to enter/exit a position is determined by generated signals. Signals are generated with the technical indicator data. The simulation system will keep track of an account balance and paid fees. Slippage will be ignored. When the simulation is finished a statistical report will be generated.

High-level overview

- The system will simulate trading on a chart dataset.
- The positions are determined with signals which are calculated from the technical indicators and appended to the chart dataset.
- The trading system has the following states: `neutral`, `long`, `short`.
- A signal can produce the following actions: `open_long`, `open_short`, `close_long`, `close_short`.
  - `open_long` will close all open short positions (if any), and open a long position.
  - `open_short` will close all open long positions (if any), and open a short position.
  - `close_long` will close the open short position.
  - `close_short` will close the open long position.
- The daily account balance is appended to the chart dataset.
  - The simulation starts with an initial account balance.

Algorithm The trading simulation system uses the following algorithm:
The program will iterate over every candle in the chart dataset until it iterated over the entire period. On each iteration the following algorithm runs:

1. Get `current` which is the current row in the chart dataset.
2. If the state is not `neutral`:
3. Update the current price for all open orders.
4. Check if the stop loss has been hit. 1. Close the position with the stop loss exit price. 1. Append the `Trade` instance to the dataframe. 1. Update account balances. 1. Set the `Trade` instance to `None`. 1. Set the state to `neutral`.

5. If the `open_long` signal is on:
6. If there is a short position open, close it.
7. Open a long position. 1. Create a new `Trade` instance. 1. Set the state to `long`.
8. If the `close_long` signal is on:
9. If there is a long position, close it. 1. Append the `Trade` instance to the dataframe. 1. Update account balances. 1. Set the `Trade` instance to `None`. 1. Set the state to `neutral`.
10. If the `open_short` signal is on:
11. If there is a long position, close it.
12. Open a short position. 1. Create a new `Trade` instance. 1. Set the state to `short`.
13. If the `close_short` signal is on:
14. If there is a short, close it. 1. Append the `Trade` instance to the dataframe. 1. Update account balances. 1. Set the `Trade` instance to `None`. 1. Set the state to `neutral`.
15. Return the trades dataframe. The account balance data is appended to the chart dataframe.

When closing positions the account balances should also be updated.
Helper functions

- Open long(contracts, price)
- Close long(price)
- Open short(price)
- Close short(contracts, price)
- Write `Trade` to dataframe
- Lot sizing
- Leverage
- Stop loss type
- Stop loss percentage
- Calculate stop loss price (also to use for trail)